IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013

Foresight and Reconsideration in Hierarchical Planning and Execution

Martin Levihn Leslie Pack Kaelbling Tomás Lozano-Pérez Mike Stilman

***Abstract*— We present a hierarchical planning and execution architecture that maintains the computational efficiency of hierarchical decomposition while improving optimality. It provides mechanisms for monitoring the belief state during execution and performing selective replanning to repair poor choices and take advantage of new opportunities. It also provides mechanisms for looking ahead into future plans to avoid making short-sighted choices. The effectiveness of this architecture is shown through comparative experiments in simulation and demonstrated on a real PR2 robot.**

I. Introduction

Our goal is to enable robots to operate in complex environments for long periods of time with substantial uncertainty in sensing and action, and fundamental lack of information about the initial state. A theoretically optimal strategy for such problems is to compute, offline, a policy that maps histories of observations into actions. This is computationally entirely intractable in large domains. A more computationally feasible strategy is interleaved online planning and execution, in which a partial plan is constructed online, using an approximate model for efficiency, the first action is executed, the belief state is updated, and a new plan is constructed. This has been demonstrated to be an effective approach in medium-sized discrete and continuous domains [1]. However, in very long-horizon problems, even finding an approximate plan in terms of primitive actions becomes computationally intractable.

Previous work [2, 3] introduced a hierarchical strategy for interleaved online planning and execution, called HPN (Hierarchical Planning in the Now), that improved computational efficiency by using a temporal hierarchical decomposition of the planning problem into many small planning problems.

In this paper we introduce two general concepts and mechanisms that can be used to improve both efficiency and optimality for integrated task and motion planning and execution in large domains: **Reconsideration** and **Foresight**. We show that the hierarchical decomposition introduced in HPN to improve efficiency provides an opportunity to apply these broader concepts to improve optimality as well, measured in the cost of actions taken by the robot. Furthermore, our new techniques allow us to demonstrate and empirically evaluate the trade-offs between efficiency and optimality in simulation and on a real robot system.

This work was supported in part by the NSF under Grants IIS-1117325 and IIS-1017076. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We also gratefully acknowledge support from ONR grant N00014-12-1-0143 and ONR MURI grant N00014-09-1-1051, from AFOSR grant FA2386-10-1-4135 and from the Singapore Ministry of Education under a grant to the Singapore-MIT International Design Center.

Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA {lpk,tlp}@csail.mit.edu; Center for Robotics and Intelligent Machines at the Georgia Institute of Technology, Atlanta, GA 30332, USA {levihn, mstilman}@gatech.edu.

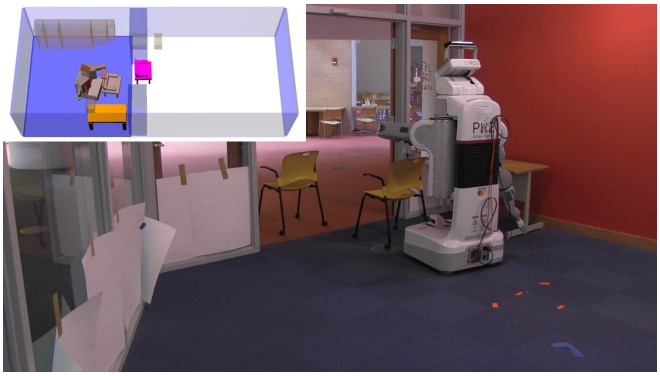

Fig. 1. Solving a NAMO problem with a PR2. The robot discovers two unknown obstacles (chairs) and moves them out of the doorway in order to get through.

Consider the following two concepts in the context of both efficiency (total computational time for all planning subproblems) and optimality (total execution cost of actions on the robot):

- **Reconsideration**: Deciding when to replan based on opportunity for plan improvement and computational cost of replanning.
- **Foresight**: Leveraging knowledge of future subgoals and beliefs about future observations to maintain correctness and improve optimality.

These concepts provide a general framework for understanding a wide variety of algorithmic strategies for improving efficiency, including the methods from the original HPN as well as a set of new methods for improving optimality that are introduced in this paper.

Reconsideration in the original HPN was used to trigger re-planning and restrict it to a subtree of the hierarchical plan; it *only* triggered when the execution system was not able to achieve the preconditions of an action. This strategy led to computational efficiency but also to considerably suboptimality due to extraneous actions. We now observe that reconsideration can also be triggered to re-plan when *new opportunities* present themselves that are likely to improve the efficiency of the overall system.

Foresight in the original HPN applied only locally: within the planning process for a particular subgoal at a particular level of abstraction, the backward chaining algorithm computed conditions characterizing correctness requirements for the later part of the plan, which were used to constrain choices made during earlier parts of a plan. This degree of foresight was critical to guarantee plan correctness and improved computational efficiency by considerably decreasing backtracking.

In this work, we show that foresight can also be applied more globally, by taking into account the robot's intentions for future action, which are encoded in the hierarchical structure of subgoals created by the HPN process. Our new system not only ensures that early action choices do not prevent future success, but also tries to miminimize the interference between separate subtasks of a plan.

This paper describes our new techniques based on the concepts of reconsideration and foresight and identifies tradeoffs between efficiency and optimality in the context of these categories of reasoning.

II. BHPN

In this section, we present an existing hierarchical planning and execution architecture, which will provide the basis for new foresight and reconsideration methods discussed in the next section.

A. Hierarchical planning and execution architecture

The BHPN (Belief HPN) architecture [3] is a hierarchical planning and execution architecture designed to work in uncertain domains. It assumes the existence of:

- A *state-estimation* module which maintains a *belief state*—a probability distribution over the states of the world—reflecting the history of observations and actions made by the robot.
- A set of *operator descriptions* characterizing the operations that the robot can do in the world; each operator description has a set of preconditions and a set of effects, described in terms of logical predicates that characterize some aspect of the robot's belief about the state of the world. Abstracted versions of the operators, obtained by postponing consideration of some of the preconditions, are used to construct *hierarchical* plans.
- A *planner* that uses the process of *pre-image backchaining*. It starts from the goal, which is a description of a set of desirable states of the world, and works backward. On each step, it computes the *pre-image* of the goal under the selected action; the pre-image is the set of states such that, were the action to be taken, then the resulting state would be in the goal set. The pre-image is represented compactly as a logical formula. A call to the planner, starting in belief state b with goal set γ results in a list $((-,g_0),(\omega_1,g_1),...,(\omega_n,g_n))$ where the ω_i are (possibly abstract) operator instances, $g_n = \gamma$, g_i is the pre-image of g_{i+1} under ω_i, and $b \in g_0$. The conditions g_i play a critical role in the execution architecture: g_i is the set of states in which the plan $\omega_{i+1},...,\omega_n$ can be expected to achieve the goal.

Figure 2 shows the BHPN algorithm. It takes as inputs a current belief state b, logical goal description γ, and a world (real robot or simulation) in which actions can be executed. It manages a stack of plans, which are stored in variable ps; it is initialized to have a dummy plan, consisting of $g_0 = True$ and $g_1 = \gamma$, indicating that it can always be applied and that its overall goal is γ. The procedure loops until the plan stack is empty, which can happen only when γ has been achieved.

On each iteration of the algorithm, we examine the plan p that is on top of the plan stack to see if it is applicable. In the following sections we will consider a different definition of the *applicable* method, but we will begin with a very simple definition of it and the related method *nextStep*. In this case, p is applicable in belief state b if the plan has not reached the end ($p.i < p.n$) and b is in the preimage $g_{p.i}$ of the next step that will be executed, which is step $i+1$. If p is not applicable, then it is popped off the stack. Note that the initial plan is always applicable.

If p is applicable, then we determine the next operator ω and subgoal γ' by calling the *nextStep* method of p. The simplest version of *nextStep* ignores the current belief state b and simply returns the sequentially next step in the plan. If the next operator is abstract ($None$ is always abstract), it means that its preconditions may not yet have been met in full detail, and so we construct a plan to achieve the subgoal γ' at a more detailed level of abstraction ($p.\alpha.incr(\omega)$) and push it on the plan stack. Otherwise, ω is a primitive, which is executed in the world, yielding an observation obs, and the belief state is updated based on that observation. After every action, we can (in principle) reconsider the plan; we discuss this in the next section.

```
BHPN(b,γ,world):
 1  ps = STACK()
 2  ps.push(PLAN Plan([(None,True),(None,γ)]))
 3  while not ps.empty():
 4      p = ps.top()
 5      if not p.applicable(b):
 6          ps.pop()
 7      else
 8          (ω,γ') = p.nextStep(b)
 9          if ω.abstract():
10              ps.push(PLAN(γ',b,p.α.incr(ω)))
11          else
12              obs = world.execute(ω)
13              b.update(ω,obs)
14              RECONSIDER(ps, b)

applicable(p,b):
1   return p.g_{p.i} = γ or (p.i < p.n and b ∈ p.g_{p.i})

nextStep(p,b):
1   if not p.g_{p.i} = γ: p.i += 1
2   return (p.ω_{p.i}, p.g_{p.i})
```

Fig. 2. Pseudo-code for the BHPN algorithm with reconsideration. In the *applicable* and *nextStep* methods we assume that the plan p has an instance variable i that is initialized to 0 and used to keep track of the current step for execution of the plan.

It is important to see that a plan p may be popped off the stack for two very different reasons: (1) All plan steps have been executed with their intended results; or (2) Some plan step, when executed, did not have its intended result. In either case, it is appropriate to return control to the higher level (the plan deeper in the stack). If the plan executed successfully, then control will resume at the next step of the plan at the higher level of abstraction; if it failed to have the desired effect, then the parent operation will also fail, and so on all the way to the bottom of the stack, where the initial plan is expanded again. Later, we consider several extensions to make this control structure more robust and flexible.

Figure 3 illustrates the nominal behavior of the BHPN algorithm, as a planning and execution tree. The blue nodes represent goals or subgoals; in this case, the overall goal, at the root node, is to have two objects in the cupboard. Goals and subgoals are characterized by conditions on belief states. The actual planning and execution tree is considerably more complicated: this example is abstracted to make it easier to explain its important aspects.

Each outlined box contains a plan. Within a single box, there are blue and pink nodes. Pink nodes represent operators and blue nodes subgoals. So, in Plan 1, using an abstract version of the *Place* operation, it is determined that, as long as both objects A and B are movable, a plan consisting of placing A, followed by placing B will achieve the goal. Note the intermediate subgoal: $\exists x.BIn(x, Cupboard) \wedge BMovable(B)$. That is the pre-image of

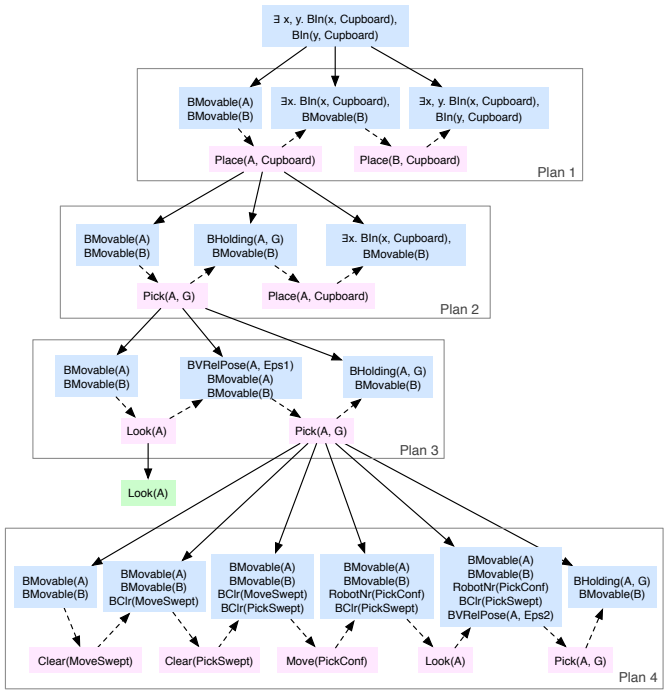

Fig. 3. Illustration of BHPN approach

the goal under the operation of placing B in the cupboard: as long as some other object is in there, and we additionally place B, then the goal will be achieved.

Plan 2 is a plan for achieving the first subgoal in Plan 1, which consists of picking up object A with grasp G and placing it in the cupboard. Plan 3 is a plan for coming to hold A in grasp G: it consists of a primitive *Look* action (shown in green), which is intended to achieve the condition $BVRelPose(A, Eps1)$, which says that we know the relative pose of the robot and object A to within some tolerance $Eps1$. This step ensures that the robot has a moderately good estimate of the pose of object A before trying to plan in detail how to pick it up. Plan 4 provides yet more detail, ensuring that the volumes of space (called "swept volumes") that the robot has to move through are clear, and that, before actually picking up the object, its pose id known to a tighter tolerance, EPS2.

The tree, as shown, is a snapshot of the BHPN process. Using the simplest *nextStep* definition, each plan would be executed left to right, completely planning for and executing actions to achieve one subgoal before going on to the next. This results in a left-to-right depth-first traversal of the tree. If any operation, primitive or abstract, has a result that is not the expected one, then it will fail and control will return to the plan above it.

B. Flexible execution of existing plans

The execution architecture described so far is very rigid: if a plan cannot be executed from left to right, then it is considered to be a failure. But, in many cases, the plan can be salvaged, and replanning is limited.

Consider Plan 4. If the *move* action doesn't have the expected result because the robot undershoots its target, for example, then a reasonable course of action would be to re-execute the move operation. The same would be true if, upon executing the *look* action, it found that the variance on its pose estimate for A was not sufficiently reduced to satisfy the preconditions for the *pick* action.

applicable(p,b):

1 **return** $b \in \bigcup_{i=0}^{n-1} p.g_i$ and $b \notin p.g_n$

nextStep(p,b):

1 $i^* = 1 + \text{argmax}_{i=0}^{n-1} b \in p.g_i$

2 **return** $(p.\omega_{i^*}, p.g_{i^*})$

Fig. 4. Pseudo-code for the *applicable* and *nextStep* methods that will re-try the current step if its preconditions still hold.

We can generalize this idea further, to handle both *disappointments*, when an action does not have its desired effect and *surprises*, when an action actually results in more useful results than expected. Figure 4 shows revised versions of *applicable* and *nextStep* that re-use a plan as much as possible. We say that a plan is applicable if any step of the plan is applicable. In other words, the current belief state b is in the pre-image of one of the operations, and the final goal has not yet been achieved. Thus, if an action results in a belief for which there is *some* action in the plan that is an appropriate next step, then the plan remains applicable and is not popped off the plan stack. In the *nextStep* method, we select the operation ω_i for execution where g_{i-1} is the "rightmost" pre-image that contains the current belief state.

In Plan 4, if the robot looks at A and finds that there is actually a previously unknown object intruding into the swept volume for the arm to pick the object, it could go back to the plan step that achieves $Clear(PickSwept)$ and continue execution from there. If it were to find, though, upon attempting to pick up A that it was, in fact, bolted to the table, the $BMovable(A)$ condition would become false. At that point, Plan 4 would be popped off the plan stack because none of the pre-images contain the current belief state. The same would be true for plans 3, 2, and 1 in succession. Replanning would be initiated for the top-level goal, hopefully finding a different object to place into the cupboard instead of A.

Another way to be flexible about the use of an existing plan is to re-order some of the steps, based on information available in the current belief state. We adopt the peephole optimization method of [4], which considers all of the plan steps with preconditions that are currently satisfied and uses a heuristic to decide which order to execute them in, or whether they are likely to interact very closely, in which case it will consider them jointly. In Plan 1, the peephole optimization may decide that it is preferable to put the larger object into the cupboard first, because both abstract actions are enabled in the initial belief state.

III. ENHANCING OPTIMALITY

The focus of the existing BHPN has been on correctness and computational efficiency often at the expense of optimality. In this section we introduce new mechanisms aimed at enhancing optimality with limited impact on efficiency.

A. Reconsideration

As we have seen, after an action is executed and the belief state is updated, it may be that the next step in an existing plan is not a good choice: it might be impossible to execute, or there might be better alternative courses of action. Replanning from scratch after every primitive operation and belief update will ensure that action choices are as up-to-date as possible. Yet, this would also be very expensive. In this section, we consider a series of approaches

that are intermediate between attempting to execute an existing plan as much as possible and replanning at every opportunity. These methods fall into two categories: flexible execution of plans that we have already made, and selective replanning. We have already seen one form of flexible plan execution implemented in the current BHPN; we now present some novel extensions.

1) Satisfaction of existing goals: We can also take advantage of *serendipity:* even though the particular subgoal, γ, that we are trying to achieve is not yet true, it might be that some subgoal in a more abstract plan (deeper in the stack) has been satisfied as a result of an unexpected action outcome or by nature. We can extend the idea of executing the rightmost executable operation in the plan at the top of the stack, and instead find the deepest (most abstract) plan in the stack that has a goal currently being expanded that is different from the rightmost goal (the final goal of the plan) at the next (more concrete) level of the stack. If there is such a plan, then we should begin execution from the next applicable step in that plan, instead. Lines 1 through 4 in Figure 5 provide pseudocode for doing this, under the assumption that we are using the definitions in Figure 4 as well.

Suppose that after carefully examining object A in the last part of Plan 4, the robot was to find that there are already two objects in the cupboard, then it would notice that the overall goal has already been achieved and return directly, rather than continuing to put A and then B in the cupboard. This is computationally inexpensive, with running time on the order of the total number of steps in all the plans on the plan stack, assuming that computing $nextStep$ for a given plan requires examining each of its pre-images.

2) Selective replanning: We might also wish to increase the robot's degree of responsiveness to change in its beliefs, while still trying to avoid unnecessary re-computation. In Figure 5, lines 5 through 9, we potentially reconsider each plan in the stack.

We therefore allow plan instances to have an optional *trigger* method: $triggerReconsideration(p, b)$. This domain-dependent function decides whether it might be advantageous to re-generate a plan for the goal that p was intended to achieve. Such triggers will be executed at every level after every action, and so care must be taken to make them computationally light-weight. They are intended to return **true** in belief states for which it is likely to be worthwhile to change plans.

If replanning is triggered, then a new plan p' is created, with the same goal and abstraction levels as for plan p, but starting from belief state b. If p' is the same as p, or a substring of p (which is a natural occurrence, since some operations from p will, in general, have already been executed), then we go on to consider the next level of the plan stack. However, if p' is different from p, then the rest of the plan stack is popped off, p' is pushed onto it, and the planning and execution process resumes.

We have experimented with two types of triggers. The first is detection of major perceptual events, such as newly detected objects, that overlap with parts of the environment that are relevant to p. The second is re-evaluation of existing plans in the new belief state, and detection of a substantial increase in cost. Although it takes time exponential in the plan length, in general, to find a plan, it is only linear time to recompute its cost starting in a new state. If we cache the cost of the plan in the belief state for which it was constructed, and find that it has become more expensive due to unexpected changes in the belief, it might warrant replanning to see if there is a better alternative.

```
RECONSIDER(ps,b):
1   for i in 0..len(ps):
2       (ω',γ') = ps_i.nextStep(b)
3       if γ' ≠ ps_{i+1}.g_{ps_{i+1}.n}:
4           ps.pop(i..len(ps))
5       if p.triggerReconsideration(b):
6           p' = PLAN Plan(p.g_{p.n},b,p.α)
7           if p' ⊈ p:
8               ps.pop(i..len(ps))
9               ps.push(p')
```

Fig. 5. Pseudo-code for handling reconsideration events. $i = 0$ is the bottom (most abstract) level of the goal stack.

To avoid extraneous computation, we might create a cascade, in which primitive perceptual events may trigger re-evaluation, which may trigger replanning, which may trigger the abandonment of the remaining planning hierarchy and adoption of a new plan. In our example, in plan 4, the robot might find that in order to clear the swept volume for picking object A, it would have to move several additional (previously unseen) objects out of the way. That information could cause the cost estimate for action of picking A in plan 2 and even for placing A in plan 1 to be higher than they were initially, and trigger replanning that might result in the use of a different grasp for A. In fact it could cause the planner to choose an entirely different object to put into the cupboard instead of A.

If we trigger replanning only when the cost of an existing plan increases, the robot will not be able to be *opportunistic*, in the sense of taking advantage of new opportunities that present themselves. For instance, it may be that the use of an object, C, was initially discounted because it was believed to be very difficult to pick and place, because it was in a different part of the room, or highly occluded by other objects. If, during the course of looking for object A, the robot were to notice that C was actually easily reachable, it might be desirable to use C instead of A in the plan. In general, detecting that a plan involving C might be better requires as much time as replanning so one can, instead, always replan after every action.

B. Foresight

We can both reduce the replanning cost and the operational cost of taking extra actions by allowing our future intentions, as encoded in the plan stack, and our future observations, as encoded in the belief state, to inform choices we make in the present. We now discuss two new strategies that we have incorporated into BHPN.

a) Constraints from Intentions: The planning problems solved by PLAN occur in the space of probability distributions over states of the robot's world. In general, the world-state space is itself effectively infinite (or, at least, unbounded finite) dimensionality, with mixed continuous and discrete dimensions. There are many ways of grasping an object or picking a specific geometric goal configuration within a target region. The planner therefore requires methods of selecting concrete variable bindings. We describe the use of *generators* in BHPN in detail in [3]. The generators use information in the current belief state and goal to select one or more reasonable values for variables. However, this process can be myopic, potentially making choices that make it more difficult to carry out future parts of the plan. To make better choices, the planner should be aware of constraints

imposed by the future parts of the plans at every level of abstraction.

This is straightforward to accomplish, structurally. We can simply pass the entire plan stack into the call to PLAN, replacing line 10 in Figure 2 with

$$ps.push(\text{PLAN}(\gamma',b,p.\alpha.incr(\omega),ps))\ .$$

For example, when clearing out the swept volume for picking up A, it might be necessary to place some objects out of the way. The intention to place A and then B into the cupboard could be taken into account when deciding where to put these other objects, so that they won't interfere with the operations of the rest of the plan. [1]

b) Predicted Observations: Another form of foresight is to avoid reconsideration or replanning in the future by taking the robot's uncertainty into account when performing geometric planning. For example, knowing that when the robot gets close to an object and looks at it, the resulting pose estimate will be drawn from the current uncertainty distribution, the planner should select paths that are less likely to be obstructed when further information is gathered. [2] This strategy can improve both efficiency and optimality: efficiency because replanning is reduced and optimality because the actions are more likely, in expectation, to be appropriate for the situation.

The mechanisms of reconsideration and foresight described in this section are applicable to most robot domains, but their particular instantiation will depend on each domain. Similarly, their effectiveness will vary among domains. In the following section, we explore the instantion and effectiveness of our mechanisms in a challenging domain.

IV. NAVIGATION AMONG UNCERTAIN MOVABLE OBSTACLES

The domain of navigation among movable obstacles (NAMO) [5] is a simply stated but challenging problem. It requires the robot to achieve a navigation goal that cannot be reached directly, but can be reached by moving obstacles out of the way. Previously, we took advantage of spatial decomposition and backward-chaining to address these problems tractably [6] and presented the first humanoid robot that performed a NAMO task [7]. More recently, NAMO research has focused on unknown [8] and uncertain [9, 10] domains, where [10] used a novel integration of Hierarchical MDPs and MCTS. While we achieved linear computation for single-obstacle displacements the policy-based approach required significant re-computation when initial object parameters proved inaccurate [11]. We therefore consider the BHPN planning approach as an alternative to policy generation.

The basic BHPN mechanisms that had previously been used in a pick-and-place domain were relatively straightforward to apply in the NAMO domain. We only had to make minor changes in the context of moving very large objects such as adjusting the path planning and object placement procedures to better model the arm and the object being held. In this section, we describe the particular methods that were used to implement foresight and reconsideration in NAMO.

[1] It is important to note that while this reasoning improves plan optimality, it is not required for correctness. During the time BHPN is performing the operation of placing A in the cupboard, it will, if necessary, move out any objects that are in the way, whether or not they were placed in the earlier part of the execution.

[2] This can be achieved by utilizing the previously introduced ϵ-shadows [3]. The ϵ-shadow of an object is defined to be the volume of space that with probability greater than $1-\epsilon$ contains all parts of the object. These ϵ-shadows can now be treated as soft constraints for a geometric path planner. A heuristic cost penalty as a function of ϵ can be applied for each initial intersection with a shadow, biasing the planner to generate plans more likely to not intersect objects.

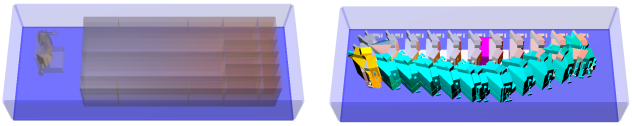

Fig. 6. Reconsideration example.

A. Implementation on PR2

We address NAMO as a mobile-manipulation domain in which a real PR2 robot must manipulate chairs in order to clear its path to reach a goal, as shown in Figure 1. The robot is given a basic map of the walls and permanent physical features of the domain, including some objects that it can use as landmarks for localization, but there are obstructing chairs that it is unaware of when it begins acting. It uses a simple object-recognition system operating on RGBD data from a Kinect sensor to detect tables and chairs in the environment.

Because the robot is real, we must take uncertainty in actions and observations seriously. We use an unscented Kalman filter to estimate relative poses among the robot and other objects in the world as well as a volumetric representation of the space that has not yet been observed [3]. The planning system will ensure that any region of space has been observed before the robot moves into it, that an object is well localized with respect to the robot before it attempts to pick it up, and that the robot is reasonably well localized with respect to the map coordinate frame before it moves the base.

The simulation results included in this paper are for a realistic simulation of the PR2, including simulated point-cloud perception and control of all degrees of freedom of the robot. It includes grasp planning, motion planning with an RRT, and view planning: the same code controls both the robot and the simulator.

B. The value of reconsideration

Exploiting serendipity (Section III-A.1) is the conceptually simplest optimization mechanism, and does not require any domain-dependent specification. It occurs most frequently in the NAMO domain when the robot performs a *look* action to determine whether some region of space is clear. In so doing, it may discover that other regions it has planned to observe in the future have already been observed in the process.

The selective replanning mechanism (Section III-A.2) requires domain dependent procedures. In the NAMO domain, we constructed the basic reconsideration trigger as follows:

1) Trigger the first stage of reconsideration if any new objects were detected that overlap any paths or placements currently being considered.
2) For any path that is under reconsideration, plan a path from the start to the goal, using new information about the domain (including known obstacles and known free space); if the new path costs less than the original path (by some fixed amount to avoid needless switching) then trigger symbolic replanning. Also, for any object placement that overlaps a new object, trigger symbolic replanning.

We also experimented with conditions in which each of the stages of reconsideration were triggered after every action: this is more computationally costly, but offers the ability to detect new opportunities that may have arisen to increase optimality.

Figure 6 shows a very simple example of reconsideration. The robot's goal is to move to the right end of the volume. As shown in the first frame, it has not yet observed most of the environment, so

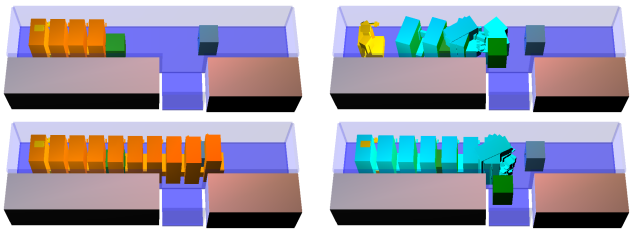

Fig. 7. Behavior with and without considering future goals when selecting placements of obstacles.

it is shrouded in *fog*. It makes a plan to move straight through to the end of the room. Then, as it looks to clear the fog, it finds an obstacle in its path. The original path and obstacle are shown in gray and magenta in the second frame. The appearance of a new object triggers reconsideration of the plan. Re-evaluation of moving along the original path reveals a much higher cost since the obstacle must be moved out of the way. Hence, replanning is triggered, resulting in a new path, shown in cyan, that avoids having to move the obstacle. In contrast, the original BHPN algorithm would have proceeded by having the robot move the obstacle out of the initially computed swept volume.

C. The value of foresight

We implemented two foresight mechanisms. The first is motivated by the insight that, in the NAMO domain, the only decisions that can make future action significantly more costly are decisions about where to place objects. Those should be made with as much information about future intentions as possible. So, the generator procedures that make geometric choices about where to place objects when moving them out of the way examine the entire plan stack, and extract all instances of predicates that constrain some region of space to be clear. The generator tries, if possible, to find placements of objects that do not intersect any of these *taboo* placement regions.

Figure 7 illustrates a domain in which there are two obstacles, and the robot must reach the end of the hallway. It decomposes the problem into two parts: clearing the left side of the path to the goal, and clearing the right side of the path to the goal. In the first row, we see the behavior without foresight: the orange shapes are approximate robot volumes indicating the swept region to be cleared. It is determined that the green box must be moved out of that volume, and so a pose just past that region is selected for the box, as shown in the second image of the first row, and the robot places the box there. While satisfying the current problem of clearing the left side of the path, this placement is little helpful in achieving the robots overall goal of reaching the end of the hallway. BHPN does recover: when it is time to clear the right part of the swept volume, both objects are successfully moved out and the goal is achieved. However, such a recovery carries the cost of moving the green box twice. In the second row, we see the same choice being made with foresight: the entire swept volume of the robot is treated as a taboo and, hence, the planner selects the *closet* at the bottom as a place for the obstacle, thus avoiding any unnecessary object motions.

BHPN has an existing mechanism for maintaining free-space connectivity that can also be seen as an example of this more general form of foresight. It operates under the assumption that the robot will always be required to move again, after the current sub-plan is finished, and so attempts to keep it as free as possible to continue. When selecting the placement of an object, the generator ensures that

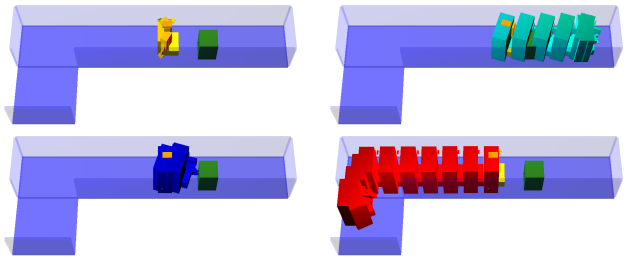

Fig. 8. Maintaining connectivity of the free space during pick and place.

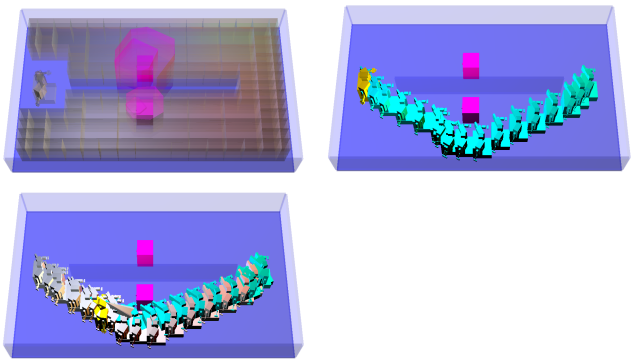

Fig. 9. Using possible future observations to inform the choice of plans.

the following paths exist: from the robot's current configuration to the configuration for picking the object, with the hand empty; from the pick configuration to the place configuration, while carrying the object; and from the place configuration back to the initial configuration, with the hand empty, *and with the object placed in the target location.* These three conditions ensure that the pick and place operations can be planned in detail and that, after placing the object, the robot will be in the same connected component of free space as it was when it started. In the current implementation, this constraint is always applied; however, there are situations in which it is important to be able to violate it (e.g., when closing a door behind you) and it will be a matter for future work to use foresight mechanisms to determine whether or not to enforce connectivity.

Figure 8 demonstrates this reasoning: there is a relatively small swept volume from the robot to its goal (shown in the second frame of the top row), but there is an obstacle in the way. It is a simple matter to move the object out of the swept volume by dragging to the left, into the hallway. However, at that point, the robot would still not be able to reach the goal. Thus, a placement for the object is generated in the bottom left of the domain, ensuring that the robot stays connected to its original pose, and is then easily able to reach the goal. In the second row, the blue swept volume is from the robot's configuration to the pick configuration; the red volume is from the place configuration back to the original configuration.

Figure 9 shows the robot using foresight to take potential future observations into account. It does so by using the current belief-state distribution on the poses of obstacles to construct "probability contours," which represent locations that might possibly be obstacles, depending on which observations are perceived. The robot needs to reach the right side of the domain, and can choose one of two hallways. Each contains an obstacle, but the pose uncertainty (shown as a lighter-colored "shadow" around the object) with respect to the obstacle in the top hallway is larger. The robot

determines that it has a better chance of getting through without having to move an obstacle if it goes through the lower hallway, even though the geometric distance is longer, and plans the path shown in the image on the right . Once it approaches and achieves a better estimate of the object location, the robot finds that it can actually take a shorter path; the bottom frame shows the original path in gray and the new one in green. In order to do this last optimization, the robot must be triggering full reconsideration after a new estimate of object location, and not relying on an increase in the estimated cost of the existing plan to trigger replanning. This type of optimization has been used in previous work; we mention it here to illustrate the ease of integrating such improvements in our framework.

V. Experiments

We compared three different versions of BHPN on different simulated domains, illustrated in Figure 10. The domains have varying number of objects, object locations as well as start and goal configurations. All the algorithms include the foresight mechanisms described earlier; they differ in their approach to reconsideration.

- **Original** BHPN: does not do any reconsideration, only replanning on failure.
- **Aggressive replanning**: replans after every action.
- **Selective replanning**: triggers replanning when new objects are discovered and when a new path to the goal is shorter than the existing plan by one meter.

For each algorithm in each domain, we measured the total simulated execution time (including all planning, simulated perception, etc.). The time column is the ratio relative to the baseline (original BHPN) times: domain 1 = 1265s, domain 2 = 1020s, domain 3 = 850s, average = 1045s. While these times also capture computations necessary for primitive action executions such as RRT calls for move actions, they do not include the time it takes to physically execute these actions by the robot. Given that the physcial execution time is typically highly system and parameter depended we instead report the number of primitive action executions. For the NAMO domain these actions are pick, place, move, and look. The results are summarized in Table I.

The results show that aggressive replanning substantially increases the runtime of our simulations compared to the original BHPN. This is due to the fact that the current plan is discarded after every action execution, independed of the outcome or obtained observations. However, as aggressive replanning always generates plans in the current belief state, it drastically reduces the number of actions that have to be executed by the robot.

The selective replanning approach attempts to merge the benefits of the original BHPN and aggressive replanning by avoiding intensive replannings as well as action executions. And indeed we observe that selective replanning improves execution optimality over the original BHPN and efficiency over aggressive replanning. The computational efficiency of selective replanning is also improved over the original BPHN as less computation for primitive actions is necessary. These results support the concepts introduced in this the paper.

To verify the applicability to very large domains we modeled our office space in simulation (Figure 10). The domain covers $100m^2$ and contains 50 objects. The robot had to navigate from the top left cubicle to the bottom right cubicle with prior knowledge of only the permanent obstacles. The robot was successfully able to reach its goal configuration while only requiring two object displacements. The right most visualization in Figure 10 shows the robots final belief state.

We also ran experiments on a real PR2, shown in Figure 1, to demonstrate the feasibility of this approach on a real robot. Again, the robot had no prior knowledge of the movable obstacles. Additionally it had substantial motion and perception uncertainty. The underlying BHPN mechanisms for robust planning and execution under uncertainty integrate effectively with reconsideration. Frequently using look actions at known landmarks to reduce positional uncertainty, the robot was successfully able to reach a goal configuration outside its current free-space region by manipulating two objects. Videos of all reported experiments as well as additional runs can be found at `http://lis.csail.mit.edu/bhpnNAMO`.

VI. Related work

There are three major threads of work related to foresight and reconsideration: one in the robotic planning literature and one in the philosophical literature of practical reasoning and the artificial intelligence literature of symbolic planning. There is a great deal of work related to integrated task and motion planning, planning under uncertainty and hierarchical planning, but we do not attempt to cover that here, see [3] for a review.

One class of algorithms specifically addresses mobile robot navigation with perfect sensing and actuation, but with fundamental initial uncertainty about the poses of obstacles in its workspace. These algorithms operate in a discretized state and action space, and employ methods based on A^* search. They assume that space is free unless it is known to be blocked, and replan when the current plan is made infeasible by newly discovered obstacles. Early versions [12] used a relatively straightforward planning strategy; more modern versions employ important algorithmic techniques to ensure computational efficiency and limit the frequency and scope of replanning. [13–15]. Likhachev *et al.* [16] integrate these methods with anytime algorithms, providing a replanning architecture that can deliver a feasible but sub-optimal plan quickly, but that converges to the optimal solution given more running time. In all of these methods, replanning is either triggered when the current plan is infeasible, or carried out as an ongoing computational process, at the highest rate possible, in response to new information gathered in real time. A successful urban autonomous vehicle [17] integrates dynamic replanning, which is triggered by plan invalidation due to a variety of environmental conditions. The work mentioned so far has concentrated on basic navigation, which is a relatively low-dimensional problem. Wu *et al.* [8] apply the ideas of cost-based reconsideration to the NAMO problem, which has dimensionality related to the number of obstacles in the domain. The PPCP method [18], like the methods described here, attempts to improve both efficiency and optimality by taking uncertainty into account when making initial plans.

Early work in symbolic planning for robots addressed issues of execution monitoring, replanning on failure, and exploiting serendipity on the Shakey [19] and Hilaire [20] robots. Recent related work in general symbolic planning solves problems with deterministic actions, but partial initial knowledge and partial sensing, by solving a sequence of classical planning problems [21, 22], with replanning triggered when the current plan becomes invalid. Similar methods can be applied in stochastic domains, through

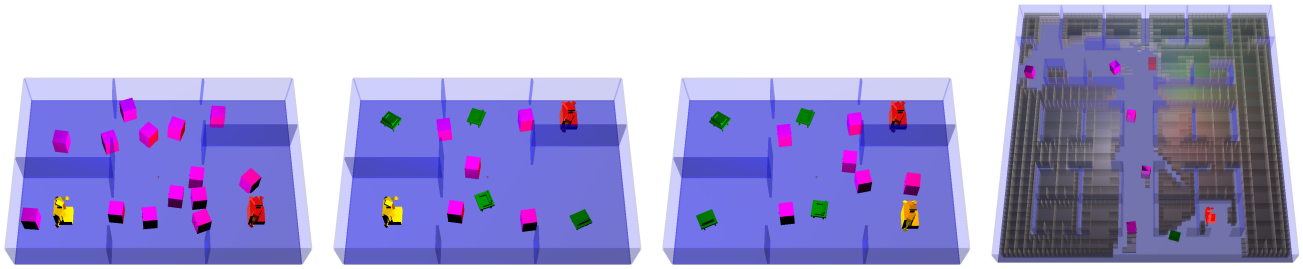

Fig. 10. Domains: The first three are used in the experiments; the robot start and goal are in orange and red, respectively. The fourth domain was used to test scalability. In all cases the robot has no initial knowledge of the objects in the environment; the gray areas in the last domain show areas that were never explored. The motion and perception uncertainty were chosen to be very low for all simulated domains.

	Original BHPN				Aggressive Replanning				Selective Replanning			
Domain	picks/places	looks	moves	time	picks/places	looks	moves	time	picks/places	looks	moves	time
1	4.75	4.25	12	1	2.1	3.2	7.8	1.01	**2**	**3**	**6.8**	**0.57**
2	3	**5**	11.8	**1**	**2**	6.3	11.1	1.66	**2**	6.3	**10.7**	1.06
3	3	6	10.3	1	**1**	5.8	8	1.40	**1**	**5**	**7**	**0.89**
Avg	3.6	5.0	11.4	1	**1.7**	5.1	9	1.33	**1.7**	**4.8**	**8.1**	**0.81**

TABLE I

RESULTS BASED ON 137 RUNS (WE PERFORMED MULTIPLE RUNS FOR EACH DOMAIN AND ALGORITHM). BEST RESULTS IN BOLD.

determinization [23]. An alternative approach is to delay decision-making until information is gathered, rather than attempting to create conditional or conformant plans in advance [24]. Fritz and McIlraith [25] consider symbolic planning in dynamic domains, including the ability to respond to changes that happen during planning.

Bratman [26] formulated a theory of rational action, which emphasizes the role of commitment to plans as a way of decreasing the burden of reasoning, and that addresses the problem reconsideration and the trade-offs it presents between efficiency and optimality. This theory was expanded into a computational architecture, IRMA [27–29] with the same high-level motivations as our work; our system can be seen as an instance of that general architecture in the domain of robotics.

Little or none of the previous work on replanning addresses many important aspects of robot planning and execution addressed here, including manipulation in high-dimensional continuous configuration spaces, control of sensing, and uncertainty in sensing action.

VII. CONCLUSION

In this paper we presented a hierarchical planning and execution architecture that incorporates concepts frequently used by humans: Reconsideration and Foresight. We showed that these concepts can be used to improve both efficiency and optimality for integrated task and motion planning and execution in large and challenging domains. The architecture enabled a real PR2 robot to solve NAMO domains with state, action and sensing uncertainty.

REFERENCES

[1] R. Platt, R. Tedrake, L. Kaelbling, and T. Lozano-Perez, "Belief space planning assuming maximum likelihood observations," in *RSS*, 2010.

[2] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *ICRA*, 2011.

[3] ——, "Integrated task and motion planning in belief space," *International Journal of Robotics Research*, 2013.

[4] D. Hadfield-Menell, L. P. Kaelbling, and T. Lozano-Perez, "Optimization in the now: Dynamic peephole optimization for hierarchical planning," in *ICRA*, 2013.

[5] M. Stilman and J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," in *Humanoids*, 2004.

[6] ——, "Planning among movable obstacles with artificial constraints," in *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, July 2006.

[7] M. Stilman, K. Nishiwaki, S. Kagami, and J. Kuffner, "Planning and executing navigation among movable obstacles," in *IROS*, 2006.

[8] H.-N. Wu, M. Levihn, and M. Stilman, "Navigation among movable obstacles in unknown environments," in *IROS*, 2010.

[9] Y. Kakiuchi, R. Ueda, K. Kobayashi, K. Okada, and M. Inaba, "Working with movable obstacles using on-line environment perception reconstruction using active sensing and color range sensor," in *IROS*, 2010.

[10] M. Levihn, J. Scholz, and M. Stilman, "Hierarchical decision theoretic planning for navigation among movable obstacles," in *WAFR*, June 2012.

[11] ——, "Planning with movable obstacles in continuous environments with uncertain dynamics," in *ICRA*, 2013.

[12] A. Zelinsky, "A mobile robot exploration algorithm," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 6, 1992.

[13] A. Stentz, "The d^* algorithm for real-time planning of optimal traverses." DTIC Document, Tech. Rep., 1994.

[14] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, 2005.

[15] S. Koenig, M. Likhachev, Y. Liu, and D. Furcy, "Incremental heuristic search in AI," *AI Magazine*, vol. 25, no. 2, p. 99, 2004.

[16] M. Lihachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," *Artificial Inteliigence*, vol. 172, no. 14, 2008.

[17] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[18] M. Likhachev and A. Stentz, "Probabilistic planning with clear preferences on missing information," *Artificial Intelligence*, vol. 173, no. 5, pp. 696–721, 2009.

[19] R. E. Fikes, P. E. Hart, and N. J. Nilsson, "Learning and executing generalized robot plans," *Artificial Intelligence*, vol. 3, 1972.

[20] R. P. Sobek and R. G. Chatila, "Integrated planning and execution control for an autonomous mobile robot," *Artificial Intelligence in Engineering*, vol. 3, no. 2, 1988.

[21] B. Bonet and H. Geffner, "Planning under partial observability by classical replanning: Theory and experiments," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2011.

[22] R. I. Brafman and G. Shani, "Replanning in domains with partial information and sensing actions," *Journal of Artificial Intelligence Research*, vol. 45, 2012.

[23] S. W. Yoon, A. Fern, and R. Givan, "FF-Replan: A baseline for probabilistic planning," in *ICAPS*, 2007.

[24] M. Eppe and D. Dietrich, "Interleaving planning and plan execution with incomplete knowledge in the event calculus," in *STAIRS*, 2012.

[25] C. Fritz and S. A. McIlraith, "Generating optimal plans in highly-dynamic domains," in *UAI*, 2009.

[26] M. Bratman, *Intention, Plans, and Practical Reason*, ser. The David Hume series. CSLI Publications, 1987.

[27] M. E. Bratman, D. J. Israel, and M. E. Pollack, "Plans and resource-bounded practical reasoning," *Computational Intelligence*, vol. 4, pp. 349–355, 1988.

[28] M. E. Pollack and M. Ringuette, "Introducing the tileworld: Experimentally evaluating agent architectures," in *AAAI*, 1990, pp. 183–189.

[29] J. F. Horty and M. E. Pollack, "Evaluating new options in the context of existing plans," *Artificial Intelligence*, vol. 127, 2001.